

Artesanato de Software: Técnicas, Refatoração e Ferramentas

Leonardo Tórtoro Pereira
leonardop@usp.br

Muito prazer!

Quem?

- Leonardo Tórtoro Pereira
- Bacharel em Ciências de Computação
 - ◆ ICMC - 2012-2016
- Mestre em Ciências de Computação e Matemática Computacional - 2017-2018
- Doutorando em CCMC 2019-202X
- Membro do FoG desde 2012
- Costuma programar em C#, Java e C

Quem?

- Pesquisador na área de Jogos
 - ◆ Foco em Geração Procedural de Conteúdo
 - ◆ Especialmente usando Algoritmos Evolutivos
 - ◆ Às vezes usando aprendizado de máquina
 - ◆ E também gramáticas formais
 - ◆ Estudando Design Patterns, Clean Code, Arquitetura de Software e tudo mais que é útil para fazer projetos grandes e com outras pessoas

Sobre este curso

Sobre este curso

- Uma visão geral sobre as áreas de estudo de boas práticas de programação nos diversos níveis de abstração
- Apresentação de algumas ferramentas que podem ajudar no desenvolvimento de códigos melhores
- Construção de um exemplo prático comparando códigos ruins com outros melhores, e aplicando refatoração

Sobre este curso

- Diversos “ponteiros” para materiais de estudo futuro para aqueles que quiserem se aprofundar
- Despertar o interesse de programar melhor, de estudar mais a forma de programar bem ao invés de aprender novas linguagens e ferramentas antes de ter boas bases
- Fazer vocês olharem para qualquer código daqui em diante e ver tantos problemas que vai fazer vocês quererem chorar

O que é artesanato de software?

Artesanato de Software

- Ênfase nas habilidades de programação
 - ◆ Código melhor ao invés de mais linhas por dia
 - ◆ Retirar a visão “de engenharia” do desenvolvimento de software
 - ◆ Foco na “arte” de desenvolver um software

Manifesto do Artesanato de Software

- Não apenas software em funcionamento,
 - ◆ Mas software de excelente qualidade
- Não apenas responder a mudanças,
 - ◆ Mas agregar valor de forma constante e crescente
- Não apenas indivíduos e suas interações,
 - ◆ Mas uma comunidade de profissionais
- Não apenas a colaboração do cliente,
 - ◆ Mas parcerias produtivas

Artesanato de Software

- De modo geral, é um movimento de programadores que tentam criar uma comunidade onde a qualidade do software criado é a melhor possível
- Mas por que?

Por que melhorar a qualidade do código?

Códigos Ruins: [1]

- Correr para liberar um produto no mercado gera código bagunçado
 - ◆ Quanto mais funcionalidades são adicionadas, pior fica o código
 - ◆ O código ruim chega ao ponto de ficar não gerenciável

Códigos Ruins: [1]

- Código ruim faz com que os programadores vaguem pelo código ao invés de programar
 - ◆ Ficar olhando pelas linhas emaranhadas e armadilhas, na esperança de alguma dica sobre o funcionamento
 - ◆ Tudo o que encontramos é mais código sem sentido

Códigos Ruins: [1]

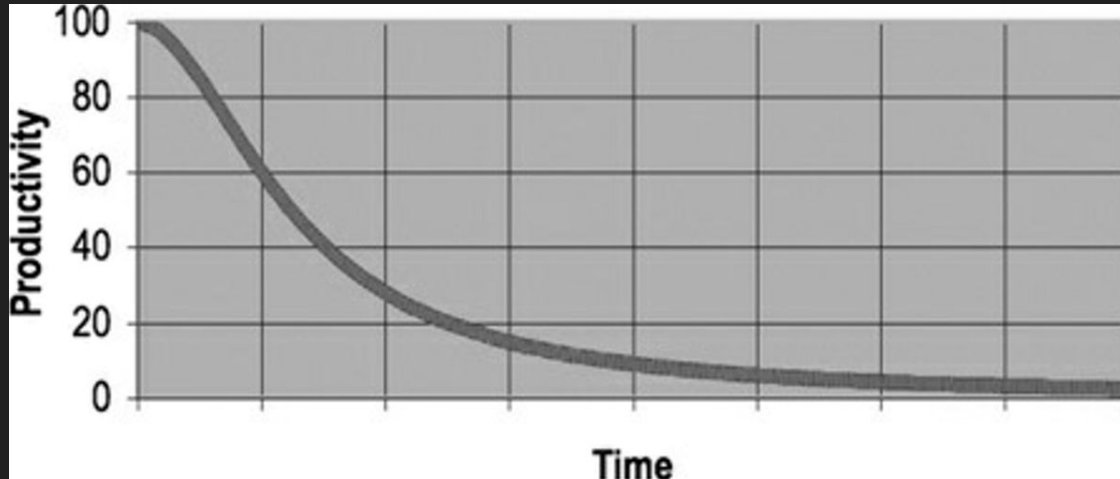
- Todos já fizemos códigos ruins. Seja por pressão de chefes, de entregas, deadlines apertadas, cansaço...
- E todos já olhamos a bagunça e pensamos que resolveríamos outro dia, pois pelo menos o código por agora funciona.
 - ◆ Nunca voltamos...

Custos de códigos Ruins [1]

- Atrasa a produção do time
- Uma mudança quebra outras 2 ou 3 partes
- Toda mudança é complexa
- Cada adição ou modificação no sistema necessita de entendimento das gambiarras pra adicionar novas gambiarras

Custos de códigos Ruins [1]

- Ao longo do tempo, a produtividade tende a 0
- ◆ Adicionar mais pessoas não é a solução!
- ◆ Os novos funcionários precisam entender a gambiarra



O que geralmente é feito?

Redesign! [1]

- Vendo a produtividade horrível, às vezes os gerentes aceitam fazer o redesign do código, mantendo um time no original.
- ◆ Com isso, tem-se dois times diferentes em códigos que fazem a mesma coisa
- ◆ Pode demorar 10 anos pra que o sistema seja reimplementado totalmente
 - E o novo sistema está tão bagunçado quanto o anterior estava

E porque isso acontece?

Quem é o culpado? [1]

- Mudança de requerimentos?
 - ◆ NÃO
- Cronogramas apertados?
 - ◆ NÃO
- Gerentes e clientes e marketeiros?
 - ◆ NÃO

A culpa é do Programador! [1]

- O seu trabalho como programador é defender o código com fervor igual aos gerentes defendem o cronograma
- Não se deve calar perante requerimentos, cronogramas e pedidos que são impossíveis de fazer a tempo.
- É preciso avisar sobre estes problemas, ajudar na construção de um cronograma plausível e que permita um bom código.

“A única maneira de cumprir o prazo (o único jeito de ir rápido) é manter o código o mais limpo possível a todo momento”

- Uncle Bob

Martin, R. C. (2009). Clean code: A handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice Hall.

Ok... Mas o que é código limpo?

Código Limpo [1]

- Não existe uma única definição correta
- Mas tende a girar em torno de boa legibilidade, modularidade e desacoplamento, além de passar em todos os testes e funcionar adequadamente
- Também pode-se dizer que é um código fácil de ler
 - ◆ “WTFs” por minuto

Código Limpo [1]

- Nós sempre lemos mais código do que escrevemos
 - ◆ Para escrever um novo módulo no sistema é preciso ler os outros e entender como funcionam
 - ◆ Para mudar o código de um sistema também
- Se você quer programar rápido, seu código precisa ser fácil de ler

E como fazer um código limpo?

E como fazer um código limpo?

- Essa é a grande questão... E envolve diversos níveis de abstração do software e, mais ainda, diversas áreas do conhecimento da computação

O que envolve um código limpo?

Tende a girar em torno de boa **legibilidade**, **modularidade** e **desacoplamento**, além de passar em todos os **testes** e **funcionar adequadamente**

O que envolve um código limpo?

Nível de Funções,
Métodos, Classes e
Design

Nível de Design e/ou
Arquitetura

Nível de Escrita
de Código

‘Mas tende a girar em torno de boa **legibilidade**,
modularidade e **desacoplamento**, além de passar em
todos os **testes** e **funcionar adequadamente**”

Nível de Testes de
Software

Nível de Engenharia de
Software/IHC/UX/Otimização

O que envolve um código limpo?

- Exceto a parte final, da funcionalidade, todos os outros termos são mais genéricos e seu aprendizado podem ser usados em qualquer projeto
- ◆ Para a funcionalidade, os conhecimentos necessário vão variar muito de projeto a projeto e qual seu papel dentro da equipe

O que envolve um código limpo?

- Mas os outros 4 são coisas que todo programador deveria buscar aprender mais e praticar, pois são as bases de um software de qualidade, independente do domínio de aplicação.
- E são eles que vamos tentar abordar, introdutoriamente, no curso

Níveis de abstração de um software [7]

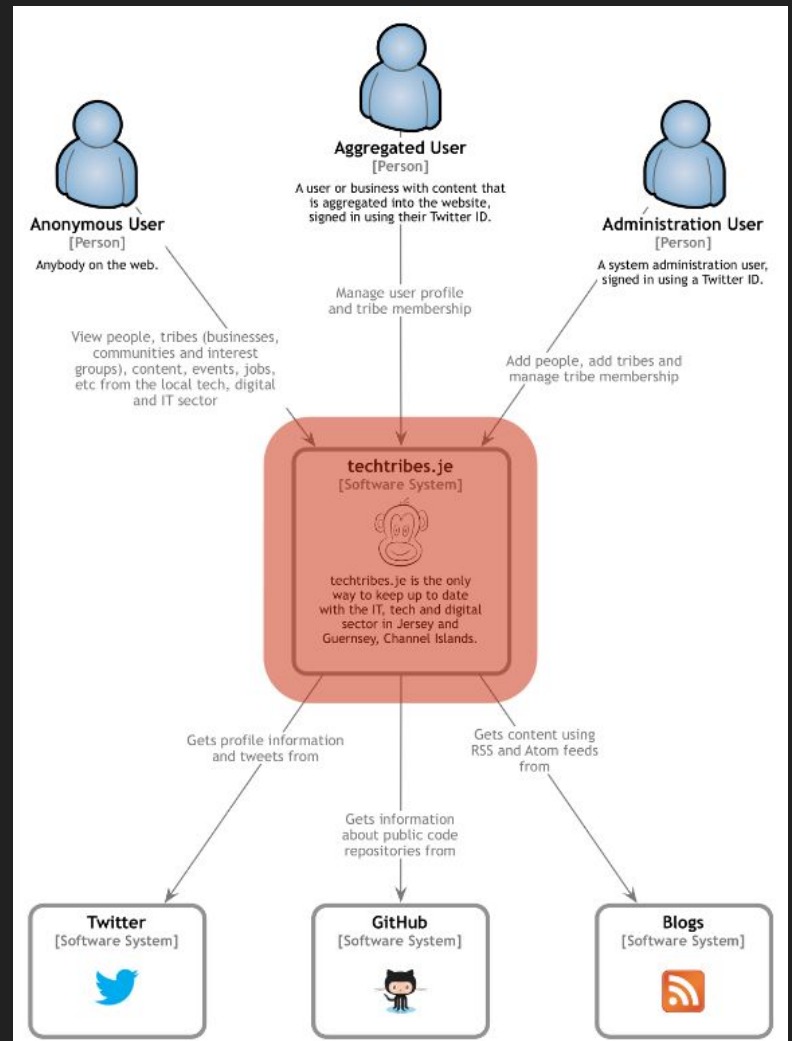
Níveis de abstração de um software [7]

- “Um sistema de software é feito de um ou mais containers, cada um com um ou mais componentes, que são implementados por uma ou mais classes”

Níveis de abstração de um software [7]

- Segundo o modelo C4 de Simon Brown
 - ◆ Um sistema pode ser construído a partir de 4 diagramas

Diagrama de Contexto



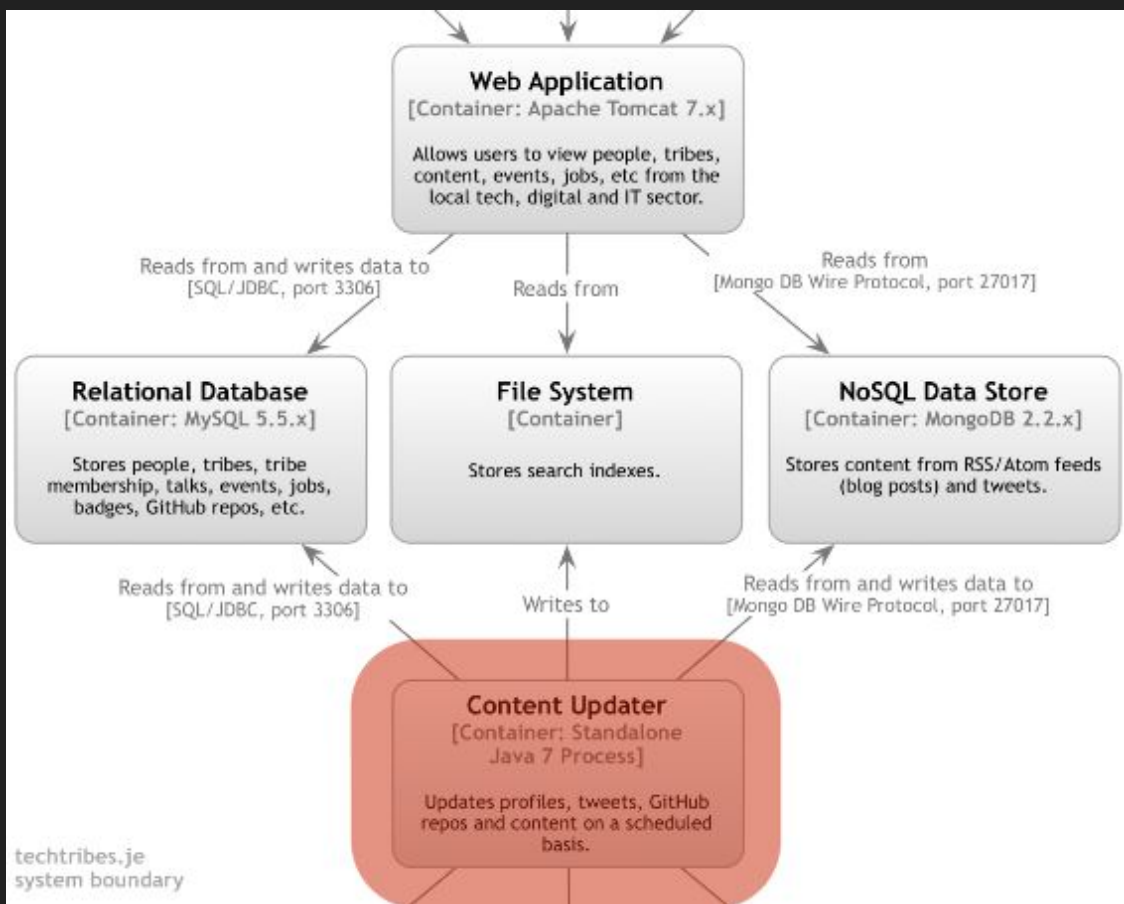


Diagrama de Containers

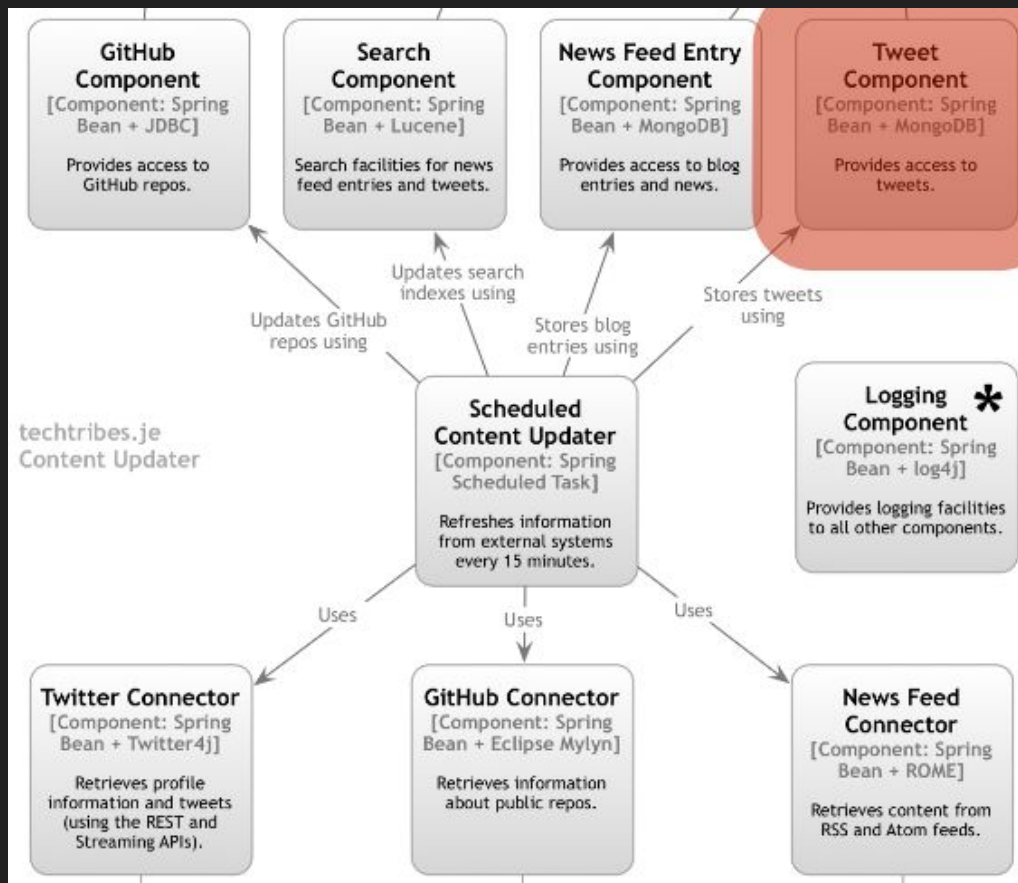


Diagrama de Componentes

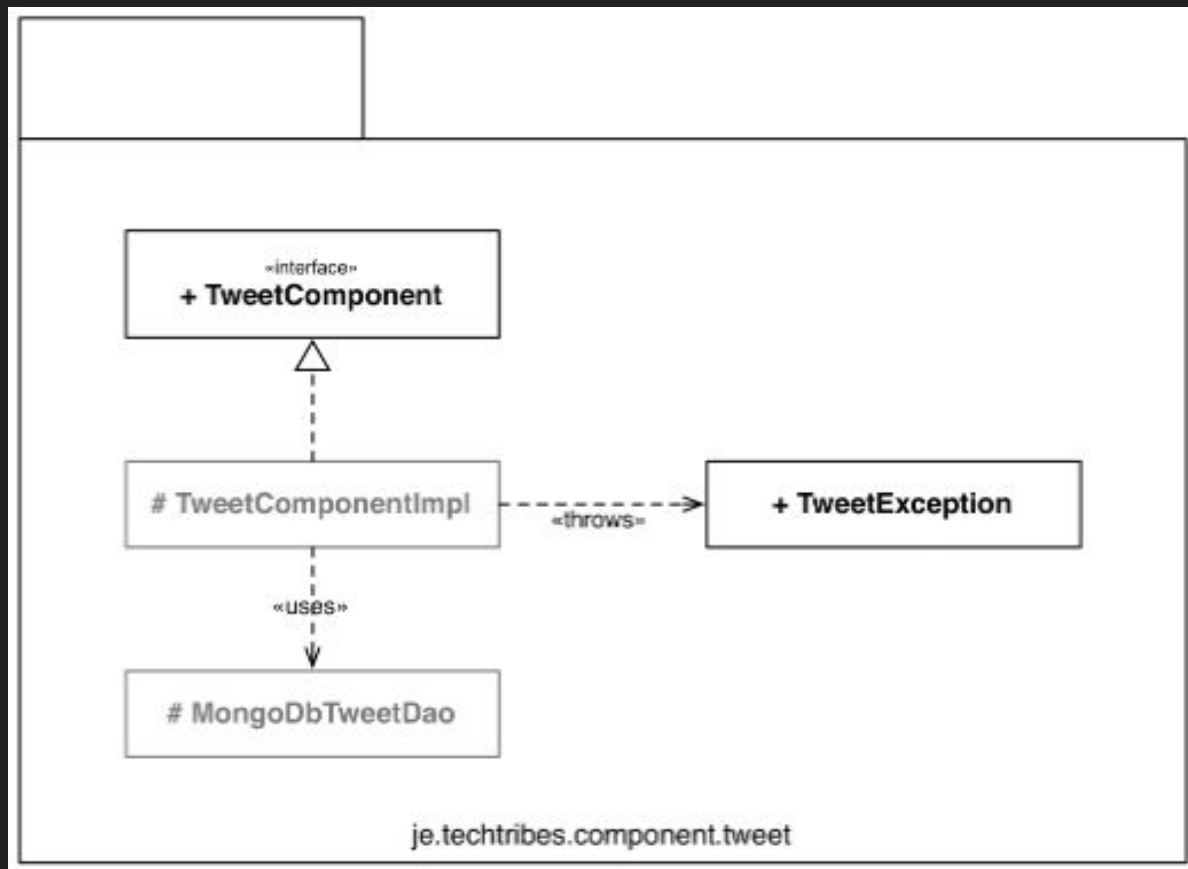


Diagrama de Classes

Componente X

Classe F



Classe G



Classe D



Classe E



Componente Y



Componente Z

Classe A

Método 1

Método 2

Método 3

Método 4

Classe B

Método 1

Método 2

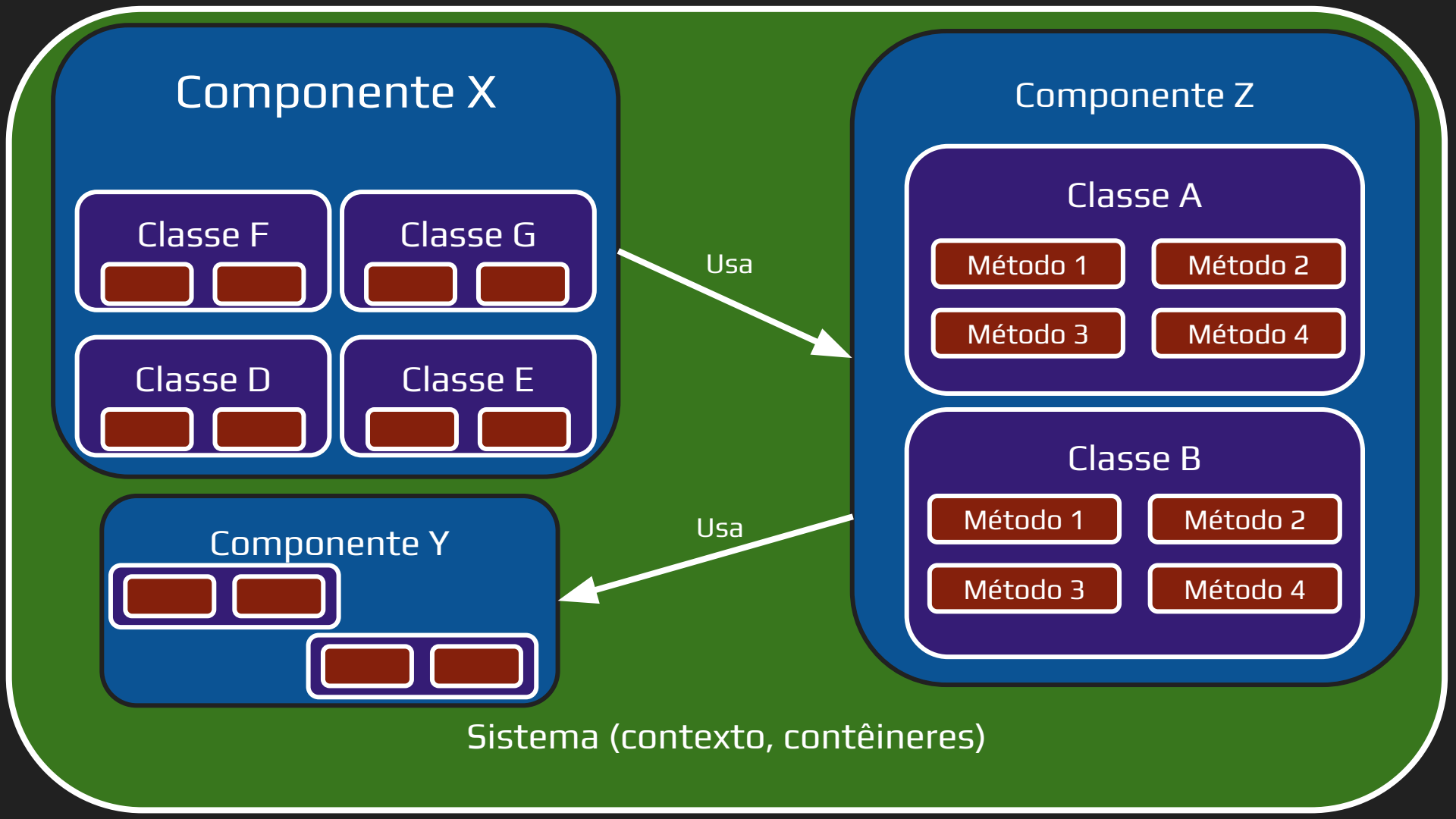
Método 3

Método 4

Usa

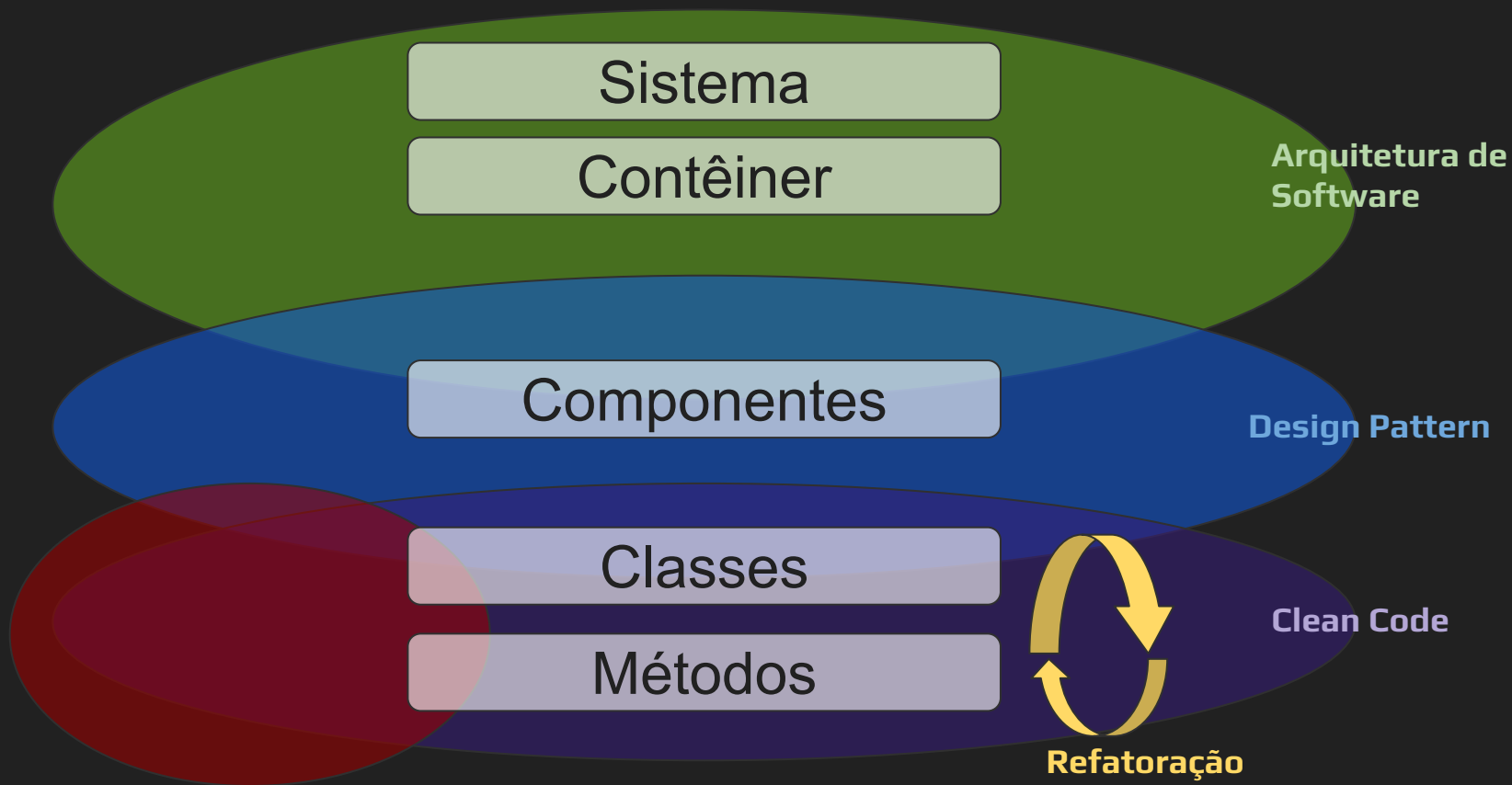
Usa

Sistema (contexto, contêineres)



E como melhorar o desenvolvimento nos diferentes níveis de abstração?

Testes de
Software



Engenharia de Software vs Arquitetura de Software [8]

Engenharia de Software vs Arquitetura [8]

→ O Engenheiro de Software

- ◆ Projeta, desenvolve e implementa soluções
- ◆ Foco nos aspectos estratégicos dos negócios
 - Maior qualidade e produtividade
- ◆ Gerenciar projetos e execuções além de programar
- ◆ Trabalha com desenvolvedores e líderes
- ◆ Métodos ágeis, teste de software, engenharia de requisitos

Engenharia de Software vs Arquitetura [8]

→ O Arquiteto de Software

- ◆ Garante que o software atenda aos requisitos
- ◆ Foco em segurança, escalabilidade e desempenho
- ◆ Decisões técnicas, resolução de problemas
- ◆ Trabalha com programadores e clientes
- ◆ Análise de arquitetura, arquitetura orientada a serviços, arquiteturas em geral

Engenharia de Software vs Arquitetura [8]

→ Em resumo:

- ◆ O engenheiro de software é focado no processo
- ◆ O arquiteto de software é focado na modelagem e projeto do produto

Arquitetura de Software

Arquitetura de Software [12]

- Tudo relacionado ao design de um sistema de software
 - ◆ A estrutura do código
 - ◆ Como o sistema funciona em alto nível
 - ◆ Como o sistema é implantado na infraestrutura

Arquitetura de Software [12]

→ O arquiteto define coisas como:

- ◆ Forma geral do sistema
 - Cliente-servidor
 - Baseado em web
 - Mobile nativo
 - Distribuído
 - Microsserviços
 - Assíncrono ou síncrono
 - ...

Arquitetura de Software [12]

- O arquiteto define coisas como:
 - ◆ A estrutura do código dentro das partes do sistema
 - Estruturado em componentes
 - Camadas
 - Features
 - Portas
 - Adaptadores

Arquitetura de Software [12]

→ O arquiteto define coisas como:

◆ A escolha de tecnologias

- Linguagem de programação
- Plataforma de implantação
- Sistemas de gerenciamento
- ...

Arquitetura de Software [12]

→ O arquiteto define coisas como:

◆ A escolha de frameworks

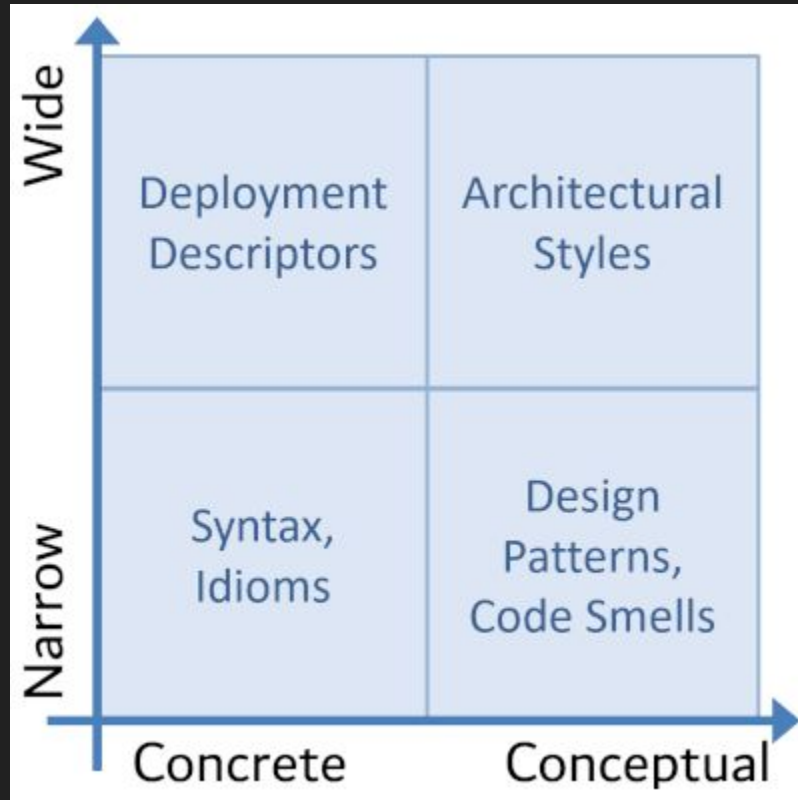
- Framework web Model-View-Control
- Framework de persistência/mapeamento objeto-relacional
- ...

Arquitetura de Software [12]

- O arquiteto define coisas como:
 - ◆ A escolha de abordagens/padrões de design
 - Abordagem com foco em performance
 - Foco em escalabilidade
 - Foco em disponibilidade
 - ...

Arquitetura de Software [12]

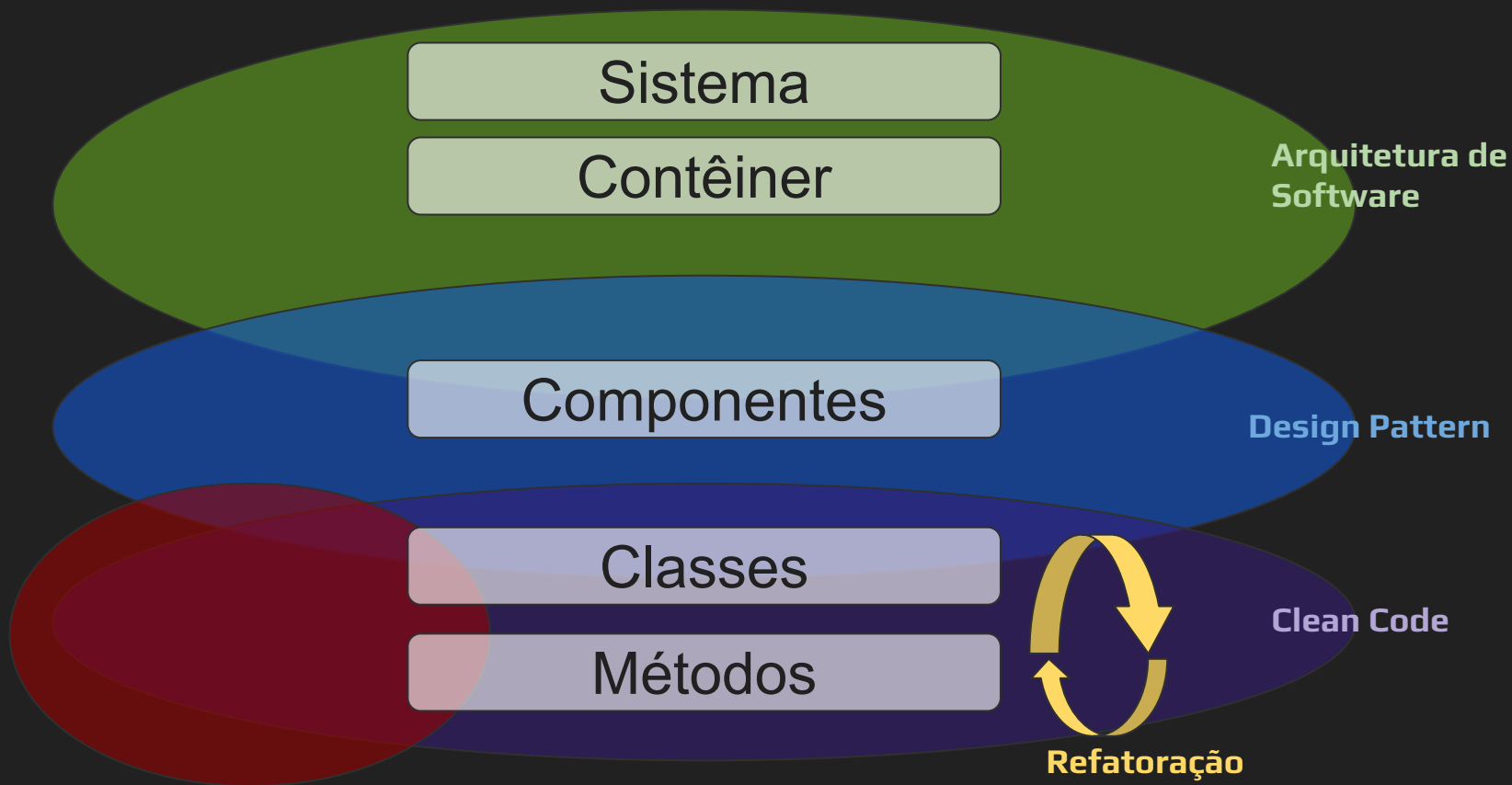
- As decisões de arquitetura são aquelas que você não consegue refatorar, e sim re-estruturar, caso mude



Divisões de preocupações arquiteturais de acordo com escopo e abstração

Fonte: [4]

Testes de
Software



Arquitetura de Software [5]

→ No caso de uma arquitetura simples, de apenas um componente de software, o arquiteto Gregor Hohpe, por exemplo, recomenda os seguintes livros:



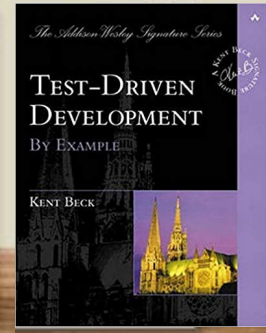
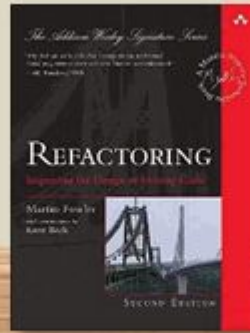
Arquitetura de Software [5]

- Esses livros, de modo geral, tratam exatamente dos escopos que mostramos no diagrama anterior!
- ◆ (com exceção de algo específico para testes)

...

Arquitetura de Software [5]

→ Esses livros, de modo geral, tratam exatamente dos escopos que mostramos no diagrama anterior!



Single Component

Arquitetura de Software [5]

- Clean Code by “Uncle” Bob Martin.
 - ◆ “Good software starts with good code, and good code is clean. Here we start with the basics of naming, functions that do one thing well, and formatting.”
- Refactoring by Martin Fowler.
 - ◆ “Good software doesn’t just happen - it evolves, gains entropy, and is then restructured. The book’s subtitle reveals that this is a book about design.”

Arquitetura de Software [5]

- Design Patterns by Gamma, Helm, Johnson, Vlissides.
 - ◆ “Going ever so slightly more abstract, design patterns help us make balanced decisions on the design of our code. This title must be close to 1/2 million copies sold by now.”

Arquitetura de Software [5]

- Pattern-oriented Software Architecture by Buschmann et al.
- ◆ “A close sibling to “GoF”, POSA 1 takes a slightly broader view, including some communication patterns. It’s the start of a whole series on software architecture patterns.”

Arquitetura de Software

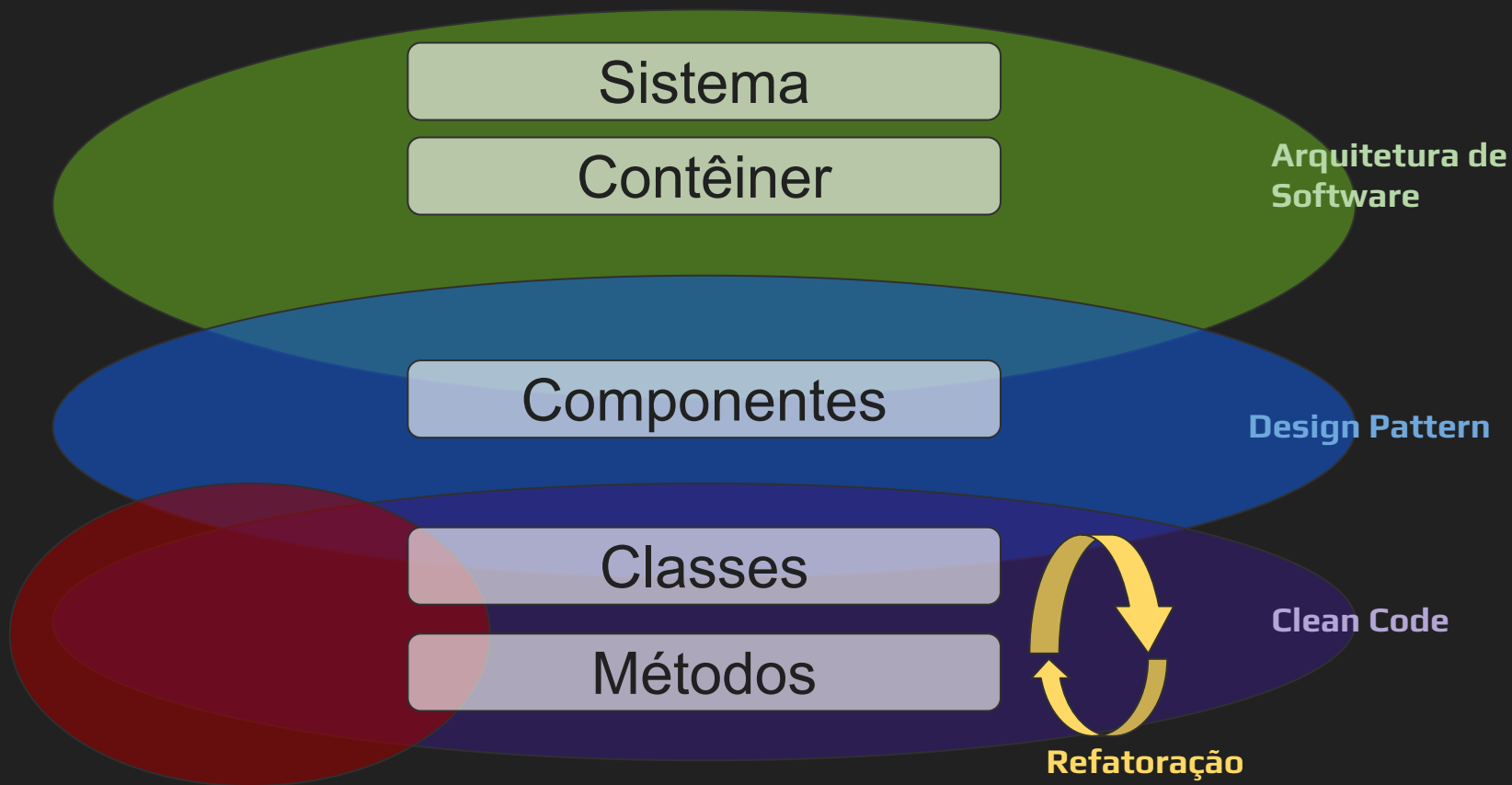
- Test-Driven Development by Kent Beck
 - ◆ Testes são parte essencial de um desenvolvimento ágil e limpo. A refatoração só é possível de ser mantida constante em um ambiente com testes.

A lista básica de um artesão completo

1. <https://www.amazon.com.br/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882>
2. <https://www.amazon.com.br/Refactoring-Improving-Design-Existing-Code/dp/0134757599>
3. <https://www.amazon.com.br/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612>
4. <https://www.amazon.com.br/Pattern-Oriented-Software-Architecture-Patterns-English-ebook/dp/B00CGH2KXA>
5. <https://www.amazon.com.br/Test-Driven-Development-Kent-Beck/dp/0321146530>

Bônus: <https://refactoring.guru/pt-br/design-patterns>

Testes de
Software



Ok, e por onde começar?

O início

- Provavelmente a melhor maneira aqui é começar bottom-up:
 - ◆ Você não vai conseguir implementar uma boa arquitetura sem um bom conhecimento de padrões de design
 - ◆ E muito menos sem programar “limpo”
 - Um código sujo é insustentável em sistemas grandes

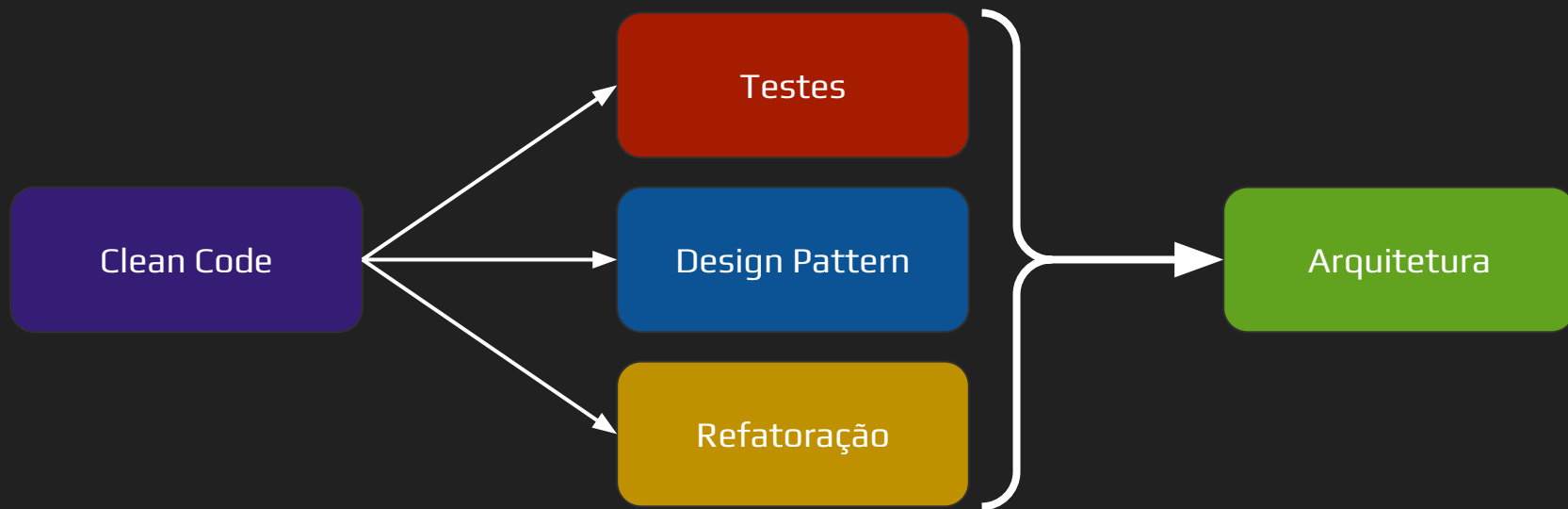
O início

- A mesma lógica vale para design pattern:
 - ◆ Se você não programar de um jeito claro, seus padrões vão ficar confusos, e o benefício deles a longo prazo deve se perder...
- Refatoração e testes envolvem ter algum código pronto para refatorar/testar...
- Portanto...

Clean Code!

A timeline de estudos

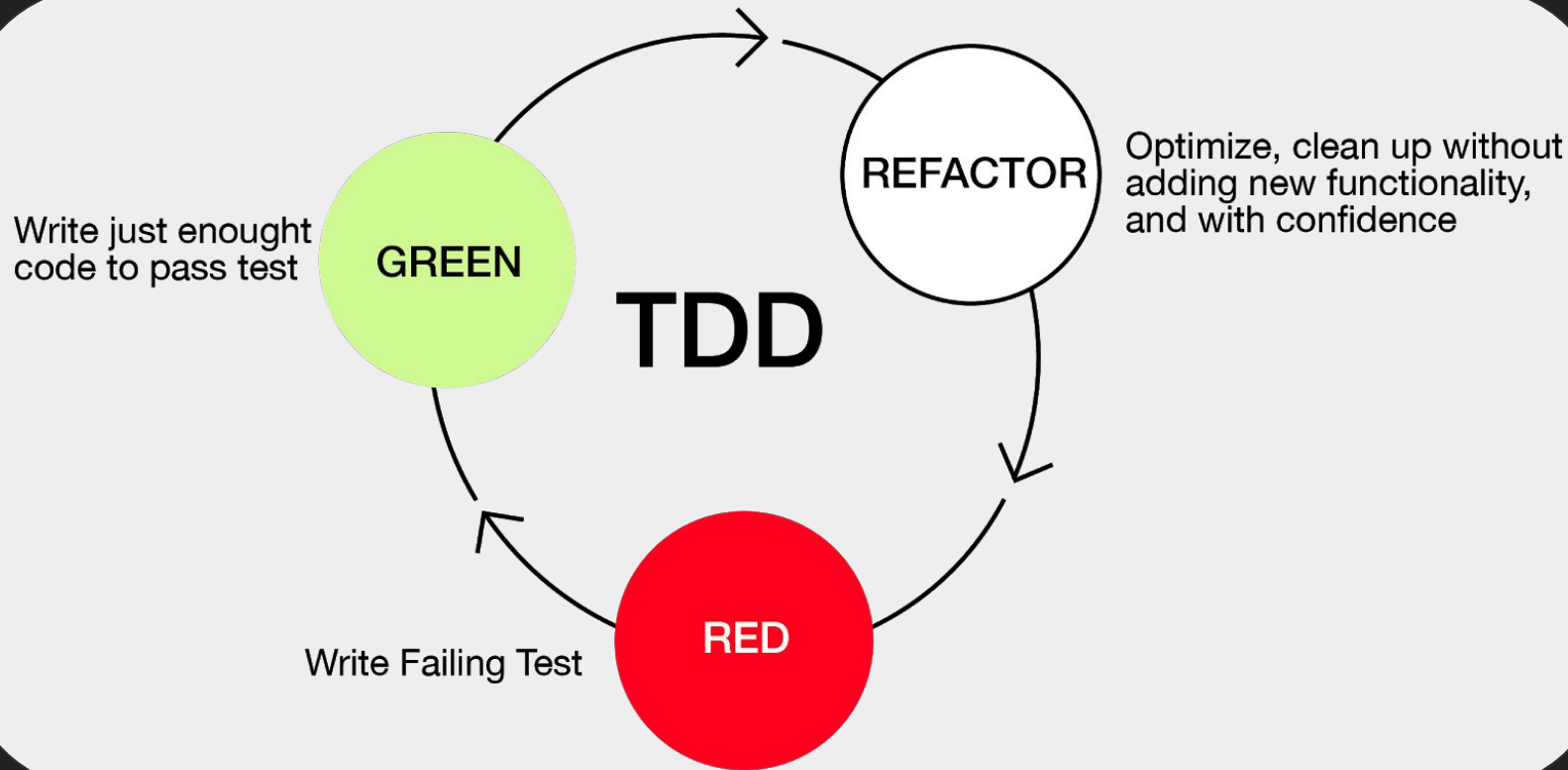
- Sim, voltamos ao tio Bob.
- Mas antes, vamos terminar a timeline:



Refatoração

Refatoração

- Mudar um código para uma versão *melhor dele
- Deve ser pequena, que pode ser feita em um commit pequeno
- Idealmente, deve ser testada por um conjunto de testes unitários e passar deles
 - ◆ Mantém o programar garantidamente funcionando
- Se for muito grande, é re-estruturação!



O ciclo da refatoração

Fonte:

<https://kislayverma.com/programming/saving-the-day-with-continuous-refactoring/>

Test Driven Development

Test Driven Development

- Desenvolver todo o código pensando em testes primeiro, funcionalidade depois
- Primeiro você programa um teste que irá verificar um pedaço bem pequeno da funcionalidade do programa
- Depois você vê que este teste falha, e adiciona a funcionalidade no código
- Ao passar, refatore
- Ao refatorar, parta para o próximo teste

Alguns guias para iniciar

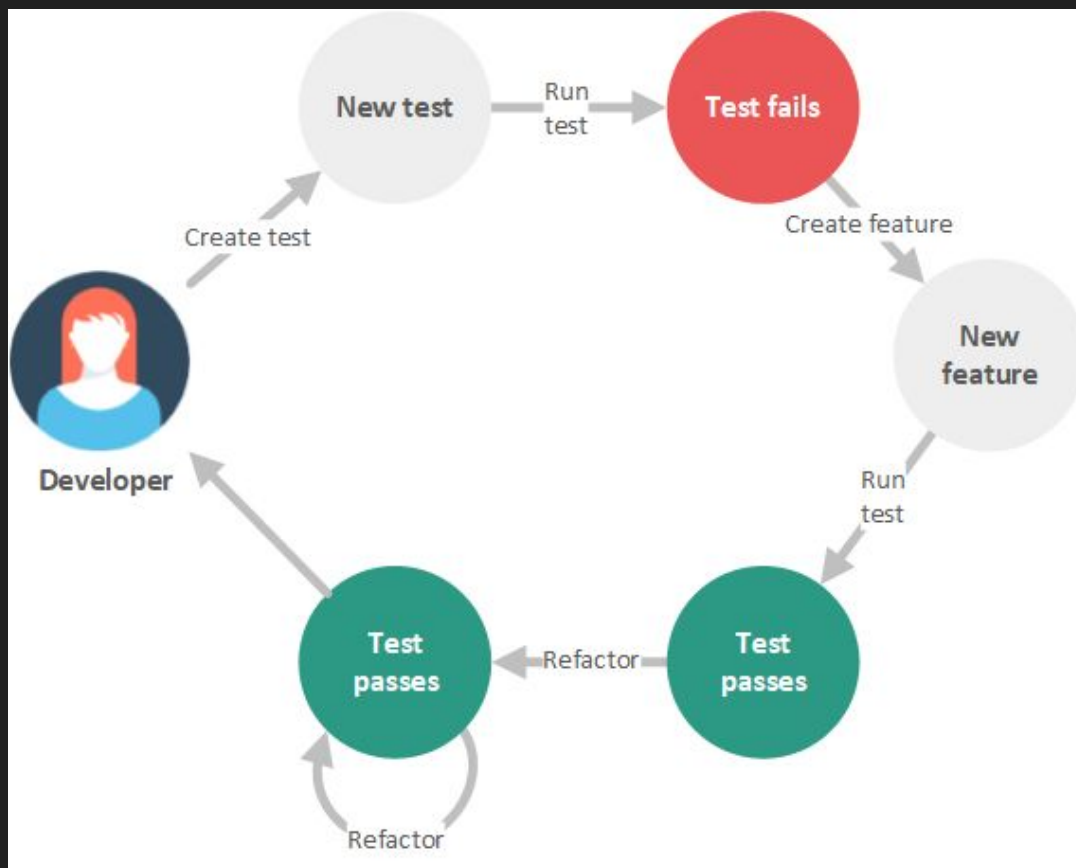
→ C#

- ◆ <https://carlosschults.net/pt/testes-unitarios-csharp-intro-tdd/>
- ◆ <https://docs.microsoft.com/pt-br/visualstudio/test/quick-start-test-driven-development-with-test-explorer?view=vs-2019>

Alguns guias para iniciar

→ Java

- ◆ <https://www.xenonstack.com/blog/test-driven-development-java/>
- ◆ <https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>



Ciclo do Test Driven Development

Fonte: <https://dzone.com/articles/what-is-refactoring>

Design Patterns

Design Patterns

- Padrões de implementação de estruturas para orientação a objetos que são muito comuns e consolidadas
- São “carimbos” que o programador pode usar para resolver elegantemente os problemas em situações que se repetem muito em quase todo software
- Focam-se em diversos conceitos de boas práticas de software
 - ◆ SOLID, GRASP, etc.

Padrões de arquitetura

Padrões de Arquitetura

- O análogo aos padrões de design, mas a um nível mais alto de abstração
- Costumam ser usados em soluções maiores, com muitos componentes

E algumas ferramentas para ajudar:

Pluigns para IDEs





- <https://www.sonarlint.org/>
- Basicamente, é um analisador estático de código que segue as regras de “smells” (mau-cheiro) do SonarSource, uma empresa aberta, com regras abertas, que ajudam a identificar problemas comuns de software

Pluigns para IDEs

- <https://www.sonarlint.org/>
- Seguir as regras que eliminam os erros que não sejam "Minor" é uma boa regra geral, pois qualquer problema de outro nível pode afetar negativamente o sistema. Já as "Minor" costumam ser detalhes que muitos consideram uma boa prática, mas dificilmente afetarão o sistema.

Pluigns para IDEs

- <https://www.sonarlint.org/>
- [Smells](#)

	 Blocker	 Critical	 Major	 Minor
Impact	High	High	Low	Low
Likelihood	High	Low	High	Low

Pluigns para IDEs

- <https://qaplug.com/>
- Plugin que ajuda na qualidade do código usando diferentes ferramentas de análise. Uma das suas vantagens é que ele busca na base de várias ferramentas ao mesmo tempo, e pode ser usado como um complemento ao SonarLint.

Solução de Servidores (Locais ou Nuvem)

- <https://www.sonarqube.org/>
- Faz uma análise mais "alta" do projeto que o SonarLint. Permite analisar a qualidade do código, segurança, débito técnico, entre outras. Tem um plano gratuito que funciona a nível mais local: pode-se criar um servidor local para analisar o projeto. Mas suas soluções pagas permitem integrar diretamente com Pull Requests em serviços de Git.

Solução de Servidores (Locais ou Nuvem)

- <https://www.sonarqube.org/>
- Para criar um servidor local é só usar o [Docker](#) e a [imagem do Sonarqube](#) dele. E então, seguir o [guia rápido](#)

Solução de Servidores (Locais ou Nuvem)

- <https://www.sonarqube.org/>
- É possível integrar ele em diversas IDEs, desde que você já tenha o servidor
 - ◆ [IntelliJ](#)
 - ◆ [Visual Studio](#)/[Outro tutorial do Visual Studio](#)
 - ◆ [Unity](#)
 - [Você provavelmente vai precisar disso pro Visual Studio/Unity](#)

Solução de Servidores (Locais ou Nuvem)

- <https://sonarcloud.io/about>
- Para códigos abertos no Github (ou privados, se você pagar), é possível usar o sonarqube na nuvem através do SonarCloud
- E ainda fazer verificações automáticas a cada push/pull request
 - ◆ E até mesmo rejeitar automaticamente os PRs que falharem

Soluções por Actions do GitHub

→ Actions do Github

- ◆ Ajudam a configurar um fluxo de trabalho de integração/implantação contínua
- ◆ Existem MUITAS actions, cada uma para uma especialidade
- ◆ No caso, podemos configurar algumas para fazer a análise estática de código (linting) direto no momento do push/pull request

Soluções por Actions do GitHub

- Algumas Actions que podem ser interessantes:
 - ◆ <https://game.ci/docs/github/getting-started>
 - Para integração contínua de jogos na Unity
 - Faz a build no servidor para testes automatizados com o Unity Test Runner

Soluções por Actions do GitHub

- Algumas Actions que podem ser interessantes:
 - ◆ <https://www.cbtnuggets.com/blog/certifications/microsoft/setting-up-a-ci-pipeline-with-github-actions-in-c-with-examples>
 - Faz a build de um projeto C# no Github e realiza testes, se existirem

Soluções por Actions do GitHub

→ Algumas Actions que podem ser interessantes:

- ◆ <https://github.com/marketplace/actions/codacy-analysis-cli>
 - Funciona similar ao Sonar Cloud: para repositórios públicos, faz uma análise de qualidade de código e você pode colocar badges no projeto
 - Também pode rejeitar PRs que não atendam critérios

Soluções por Actions do GitHub

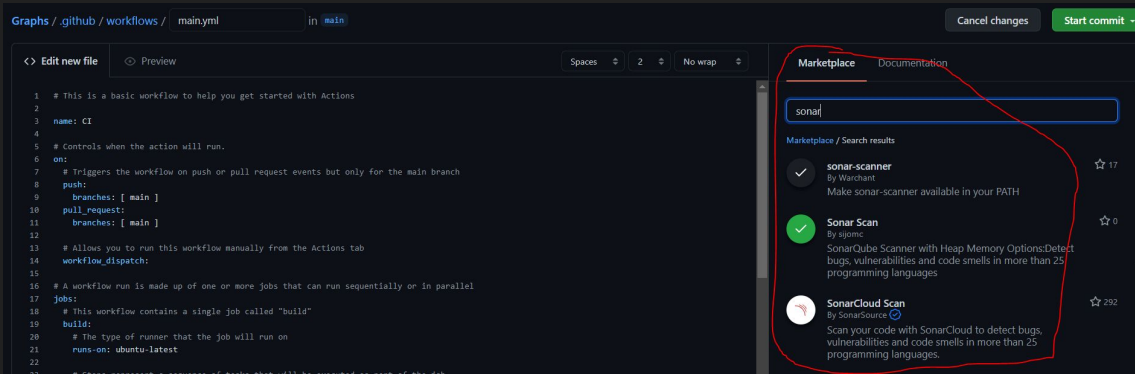
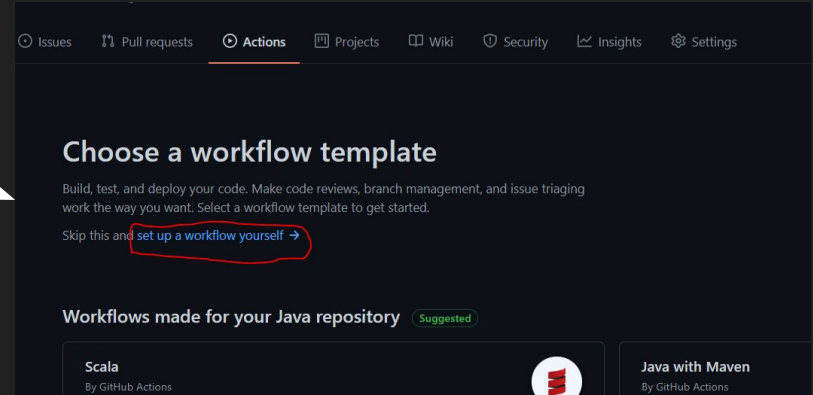
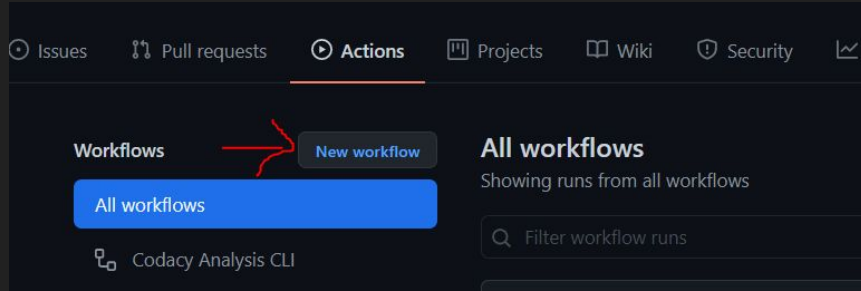
→ Algumas Actions que podem ser interessantes:

- ◆ <https://github.com/marketplace/actions/super-linter>
 - Roda linters de várias linguagens automaticamente
 - Também pode ser configurado para rejeitar automaticamente PRs que não passem neles

Soluções por Actions do GitHub

- Para procurar mais Actions, você pode ir para o Marketplace do GitHub
 - ◆ <https://github.com/marketplace>
 - ◆ Ou adicionar direto no repositório

Soluções por Actions do GitHub



Soluções por Actions do GitHub

- É possível configurar as Actions para funcionarem apenas nos códigos novos, adicionados/modificados pelo commit específico
- ◆ <https://github.com/codeocelot/git-diff-lint>
- ◆ https://medium.com/@joey_9999/how-to-only-lint-files-a-git-pull-request-modifies-3f02254ec5e0

Pull Request?

<https://yangsu.github.io/pull-request-tutorial/>

Ah, e deixe o repositório bem
documentado:

<https://dev.to/reginadiana/como-escrever-um-readme-md-sensacional-no-github-4509>

Ok, agora de volta ao Clean Code

Clean Code [1]

- De modo geral, são um conjunto de regras para deixar seu código mais claro
- O foco é num código fácil de ler
- Com nomes intuitivos
- Funções curtas e com escopo bem claro
- Sem comentários
- Boa formatação

Clean Code [1]

- Bom uso de orientação a objetos (encapsulamento, herança, polimorfismo, etc.)
- Boa manipulação de erros
- Boas práticas para uso de códigos externos
- Uso de testes unitários
- Classes bem organizadas
- Uso de padrões de projetos
- Dicas para lidar com concorrência

Clean Code [1]

- Refatoração
- Smells
- ...

É muita coisa!

Prática

- Por isso, vamos tentar começar com um exemplo
- Vamos montar uma classe de Grafos, com algumas funcionalidades, em C#
- Primeiro, vamos ver soluções ruins.
- E depois, fazer uma melhor passo a passo

Mãos à obra!

Referências

1. Martin, R. C. (2009). Clean code: A handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice Hall.
2. Fowler, M. (2019). Refactoring: Improving the design of existing code.
3. Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston, MA: Prentice Hall. ISBN: 978-0-13-449416-6
4. <https://architectelevador.com/architecture/architect-path/>
5. <https://architectelevador.com/architecture/architect-bookshelf/>
6. <https://www.slideshare.net/theojungeblut/2013-106-clean-code-part-i-design-patterns>
7. <https://herbertograca.com/2015/11/10/software-architecture-vs-code-by-simon-brown/>
8. <https://www.treinaweb.com.br/blog/voce-sabe-a-diferenca-entre-um-engenheiro-e-um-arquiteto-de-software>
9. <https://www.devmedia.com.br/arquitetura-de-software-desenvolvimento-orientado-para-arquitetura/8033>
10. <https://www.fullstacktutorials.com/architectural-patterns-vs-design-patterns-57.html>
11. <https://refactoring.guru/pt-br/design-patterns/catalog>
12. <https://leanpub.com/software-architecture-for-developers/read#c4>

Livros/Sites que podem ajudar

- Olhem os links do Gregor Hohpe
 - ◆ <https://architectelevators.com/architecture/architect-bookshelf/>
- <https://gameprogrammingpatterns.com/>